**Homework: Code Tuning and Optimization**

This document defines the homework assignments from the ["High Quality Code" HYPERLINK "http://softuni.bg/courses/high-quality-code/" Course @ Software University](http://softuni.bg/courses/high-quality-code/). Please submit as homework a single **zip** / **rar** / **7z** archive holding the solutions (source code) of all below described problems.

* **Clean the Smelly Code**

You are given a C# application ([Code-Tuning-and-Optimization-Homework.zip](10.%20Code-Tuning-and-Optimization-Homework.zip)) which displays an animated 3D model of the Solar system.

* Use a profiler to find the places in its source code which cause significant performance degradation (bottlenecks).
* Provide a screenshot of the profiler’s result and indicate the place in the source code where the bottleneck resides (name of the file, line of code).
* Make a quick fix in the source code in order to significantly improve the performance. Test the code after the fix for correctness + performance.
* **Performance of operations**
* Write a program to compare the performance of **add, subtract, increment, multiply, divide** for **int, long, float, double and decimal** values.
* Write a program to compare the performance of **square root, natural logarithm, sine** for **float**, **double** and **decimal** values.

Perform the tests **many times** (say, 100, 500 or 1000 times) **with the same parameters**, then average the times and put them in the table below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **n = 500** | **int** | **long** | **double** | **decimal** |
| **+** | 00.0000014 | 00.0000013 | 00.0000012 | 00.0000059 |
| **-** | 00.0000013 | 00.0000015 | 00.0000012 | 00.0000070 |
| **++ (prefix)** | 00.0000017 | 00.0000007 | 00.0000013 | 00.0000011 |
| **++ (postfix)** | 00.0000008 | 00.0000011 | 00.0000012 | 00.0000012 |
| **+= 1** | 00.0000017 | 00.0000016 | 00.0000017 | 00.0000015 |
| **\*** | 00.0000012 | 00.0000012 | 00.0000012 | 00.0000012 |
| **/** | 00.0000017 | 00.0000015 | 00.0000012 | 00.0000019 |

|  |  |  |
| --- | --- | --- |
| **n = 500** | **double** | **decimal** |
| **Math.Sqrt()** | 00.0000025 | 00.0001902 |
| **Math.Log()** | 00.0000055 | 00.0002206 |
| **Math.Sin()** | 00.0000025 | 00.0002050 |

For this problem, you will need to submit the two tables in a text file along with your program. Here is a tool which can create ASCII tables: <http://ozh.github.io/ascii-tables/>.

* **\* Compare Sorting Algorithms**

Write (or copy from the Internet) some implementations of sorting: insertion sort, selection sort, merge sort, quick sort. Compare their performance. You can look at the **System.Diagnostics.Stopwatch** class for a way to calculate the time a method takes to run.

Pass a few parameters and see how long it takes for the method to finish. Fill in the table below. Write "hangs" if the execution does not finish within 45-60 seconds.

You may especially check the following cases (If you check them, you will need to create more rows in your table):

* Random values
* Values in reversed order
* Many repeating values

You can optionally pass some more intermediate values and make a plot: place the parameter on the X axis, and the execution time on the Y axis.

For this problem, you will need to submit the table in a text file along with your program.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Method** | **n = 10** | **n = 50** | **n = 100** | **n = 1000** | **n = 10 000** | **n = 100 000** | **n = 1 000 000** | **n = 10 000 000** |
| **Insertion** |  |  |  |  |  |  |  |  |
| **Selection** |  |  |  |  |  |  |  |  |
| **Merge** |  |  |  |  |  |  |  |  |
| **Quick** |  |  |  |  |  |  |  |  |

* **\* Compare Data Structures**

Write a phone book application, containing people **names** and **phones**. Use two approaches: using a **List<Person>** (string name, string phone) and a **Dictionary<string, string>** (key: name, value: phone).

Write a method to search for a given person's phone. Using the list, you have to search in the entire list. Using the dictionary, you can select the index directly.

Generate a lot of people and perform **a lot of searches** (n = number of calls to the **Search(string personName)** method). See how long it takes for the method to finish. Fill in the table below. Write "hangs" if the execution does not finish within 45-60 seconds.

You can optionally pass some more intermediate values and make a plot: place the parameter on the X axis, and the execution time on the Y axis.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Approach** | **n = 10** | **n = 50** | **n = 100** | **n = 1000** | **n = 10 000** | **n = 100 000** | **n = 1 000 000** | **n = 10 000 000** |
| **List** |  |  |  |  |  |  |  |  |
| **Dictionary** |  |  |  |  |  |  |  |  |

Make a second table, where **the number of searches is constant** (say, 1000), and n = number of people. See how the results differ.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Approach** | **n = 10** | **n = 50** | **n = 100** | **n = 1000** | **n = 10 000** | **n = 100 000** | **n = 1 000 000** | **n = 10 000 000** |
| **List** |  |  |  |  |  |  |  |  |
| **Dictionary** |  |  |  |  |  |  |  |  |

For this problem, you will need to submit the two tables in a text file along with your program.